UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/720,726	11/24/2003	Adam G. Wolff	LZLO-01006US0	2756

28554 7590 03/03/2008

VIERRA MAGEN MARCUS & DENIRO LLP
575 MARKET STREET SUITE 2500
SAN FRANCISCO, CA 94105

EXAMINER	
CHEN, QING	

ART UNIT	PAPER NUMBER
2191	

MAIL DATE	DELIVERY MODE
03/03/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/720,726	WOLFF ET AL.	
	Examiner	Art Unit	
	Qing Chen	2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *26 December 2007*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-17,19-31,33-36,38-58 and 60-65* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-17,19-31,33-36,38-58 and 60-65* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *16 April 2004* is/are: a)☐ accepted or b)☒ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This Office action is in response to the amendment filed on December 26, 2007.

2. **Claims 1-17, 19-31, 33-36, 38-58, and 60-65** are pending.

3. **Claims 1-3, 5, 6, 8-10, 12, 13, 16, 17, 20, 21, 23, 24, 26, 33, 35, 36, 38, 42, 43, 45, 46, 49, 52, 58, 62, 63, and 65** have been amended.

4. **Claims 18, 32, 37, and 59** have been cancelled.

5. The objection to the oath/declaration is withdrawn in view of Applicant's submission of the Application Data Sheet.

6. The objections to the drawings are withdrawn in view of Applicant's amendments to the drawings and the specification.

7. The objection to the abstract is withdrawn in view of Applicant's amendments to the abstract.

8. The objections to the specification due to informalities are withdrawn in view of Applicant's amendments to the specification. However, Applicant's amendments to the specification fail to address the objections due to ineffective attempt of incorporating subject matter by reference and the use of trademarks. Accordingly, these objections are maintained and further explained below.

9. The objections to Claims 1-5, 8, 9, 12-16, 21, 23, 24, 32-37, 43, 45, 46, 52, and 63 are withdrawn in view of Applicant's amendments to the claims or cancellation of the claims.

10. The 35 U.S.C. § 112, second paragraph, rejections of Claims 5, 9, 10, 13-15, 17, 18, 21-24, 26, 29-32, 35-47, 50, 51, and 65 are withdrawn in view of Applicant's amendments to the claims or cancellation of the claims. However, Applicant's amendments to the claims fail to

address the rejections of Claims 48 and 49 due to insufficient antecedent basis. Accordingly, these rejections are maintained and further explained below.

11. For clarity of the record, the Preliminary Amendment (received on 04/16/2004) was entered and considered.

12. It is noted that the Amendments to the Specification contains the following typographical errors:

- The replacement paragraph [0040] should be [0041].
- "Please replace paragraph [0054] as follows:" should read -- Please replace paragraph [0053] as follows: --.
- The replacement paragraph [0058] should be [0059].

13. It is noted that Claim 38 contains proposed amendments. However, the text of the added subject matter is not shown by underlining the added text.

Response to Amendment

Drawings

14. The drawings in response to the pre-exam formalities notice were received on April 16, 2004. These drawings are not acceptable because the drawings are not in compliance with 37 CFR § 1.121(d). Any changes to an application drawing must be in compliance with 37 CFR § 1.84 and must be submitted on a replacement sheet of drawings, which shall be an attachment to the amendment document and, in the top margin, labeled "Replacement Sheet." Since Applicant has already submitted replacement drawing sheets 1 and 2 for correcting the drawing informalities, Applicant needs to submit replacement drawing sheets 3-7 with "Replacement Sheet" labeled in the top margin.

Specification

15. The attempt to incorporate subject matter into this application by reference to copending application no. 10/092,010 is ineffective because the root words "incorporate" and/or "reference" have been omitted. See 37 CFR 1.57(b)(1).

The incorporation by reference will not be effective until correction is made to comply with 37 CFR 1.57(b), (c), or (d). If the incorporated material is relied upon to meet any outstanding objection, rejection, or other requirement imposed by the Office, the correction must be made within any time period set by the Office for responding to the objection, rejection, or other requirement for the incorporation to be effective. Compliance will not be held in abeyance with respect to responding to the objection, rejection, or other requirement for the incorporation to be effective. In no case may the correction be made later than the close of prosecution as defined in 37 CFR 1.114(b), or abandonment of the application, whichever occurs earlier.

Any correction inserting material by amendment that was previously incorporated by reference must be accompanied by a statement that the material being inserted is the material incorporated by reference and the amendment contains no new matter. See 37 CFR 1.57(f).

16. The disclosure is objected to because of the following informalities:

- "webserver" should read -- web server -- on page 12, paragraph [0042].

Appropriate correction is required.

17. The use of trademarks, such as WINDOWS and JAVASCRIPT, has been noted in this application. Trademarks should be capitalized wherever they appear (capitalize each letter OR

accompany each trademark with an appropriate designation symbol, *e.g.*, ™ or ®) and be accompanied by the generic terminology (use trademarks as adjectives modifying a descriptive noun, *e.g.*, "the JAVA programming language").

Although the use of trademarks is permissible in patent applications, the proprietary nature of the marks should be respected and every effort made to prevent their use in any manner, which might adversely affect their validity as trademarks.

Claim Objections

18. **Claims 16 and 40** are objected to because of the following informalities:

- **Claim 16** recites the limitation "the new object code file." Applicant is advised to change this limitation to read "the optimized object code file" for the purpose of providing it with proper explicit antecedent basis.
- **Claim 40** recites the limitation "the application." Applicant is advised to change this limitation to read "the computer application" for the purpose of providing it with proper explicit antecedent basis.

Appropriate correction is required.

Claim Rejections - 35 USC § 112

19. The following is a quotation of the second paragraph of 35 U.S.C. 112:

> The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

20. **Claims 1-17, 38-58, and 60-65** are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Claims 1, 38, 52, and 62 recite the limitation "the initialization code." There is insufficient antecedent basis for this limitation in the claims. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "an initialization code" for the purpose of further examination.

Claims 1, 38, 52, 58, and 62 recite the limitation "the application source code." There is insufficient antecedent basis for this limitation in the claims. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "an application source code" for the purpose of further examination.

Claims 1-17 and 19 depend on Claim 1 and, therefore, suffer the same deficiencies as Claim 1.

Claims 39-51 depend on Claim 38 and, therefore, suffer the same deficiencies as Claim 38.

Claims 53-57 depend on Claim 52 and, therefore, suffer the same deficiencies as Claim 52.

Claims 60 and 61 depend on Claim 58 and, therefore, suffer the same deficiency as Claim 58.

Claims 63-65 depend on Claim 62 and, therefore, suffer the same deficiencies as Claim 62.

Claim 38 recites the limitations "the compiled object code" and "the computer application." There are insufficient antecedent bases for these limitations in the claim. In the interest of compact prosecution, the Examiner subsequently interprets these limitations as reading "a compiled object code" and "a computer application," respectively, for the purpose of further examination.

Claims 39-51 depend on Claim 38 and, therefore, suffer the same deficiencies as Claim 38.

Claims 48 and 49 recite the limitation "the application." There is insufficient antecedent basis for this limitation in the claims. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "the computer application" for the purpose of further examination.

Claims 50 and 51 depend on Claim 49 and, therefore, suffer the same deficiency as Claim 49.

Claim Rejections - 35 USC § 102

21. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

22. **Claims 20-23, 25, 27-31, 33, and 36** are rejected under 35 U.S.C. 102(e) as being anticipated by **US 7,191,441 (hereinafter "Abbott")**.

As per **Claim 20**, Abbott discloses:

- compiling an application provided in a source language *(see Column 8: 3-5, "The Java VM further includes a just-in-time (JIT) compiler 190. This forms machine code to run directly on the native platform by a compilation process from the class files.")*;
- initializing the application in a runtime environment *(see Column 9: 55-58, "Thus the method commences with the start of the Java application whose state is to be saved (step 410), which in turn leads in standard fashion to the initialisation of the Java VM classes (step 420).")*; and
- creating a serialized representation of the application *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 9: 10-20, "... the preferred embodiment of the present invention provides a set of callable routines and tools that allow a*

user to store away the state of an initialised Java application. The stored application is available for subsequent reloading, but without the start-up cost of initialisation.").

As per **Claim 21**, the rejection of **Claim 20** is incorporated; and Abbott further discloses:

- reading from the runtime environment a description of each object of the application *(see Column 12: 6-10, "Thus the first item is to create a vector or table 510 having three columns. Each object in turn in the heap is then identified from the system Reference queue, and added into the table 510, with one object entry per row of the table.").*

As per **Claim 22**, the rejection of **Claim 21** is incorporated; and Abbott further discloses:

- outputting the description to a rebuilder *(see Column 18: 10-14, "... the saved state is loaded by a special "javaload" tool, which loads the stored application into a Java VM, bypassing all the initialisation normally associated with bringing up a Java VM. The reloading of the application is performed in stages.").*

As per **Claim 23**, the rejection of **Claim 22** is incorporated; and Abbott further discloses:

- storing the serialized representation in a text file and providing the text file to the rebuilder *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)." and 55-57, "A particular benefit of this approach is where multiple, identical, VMs can all be rapidly launched from a single save file (text file)."; Column 18: 10-14, "... the saved state is loaded by a special "javaload" tool, which loads the stored application into a Java VM, bypassing all the*

initialisation normally associated with bringing up a Java VM. The reloading of the application is performed in stages.").

As per **Claim 25**, the rejection of **Claim 20** is incorporated; and Abbott further discloses:

- wherein the runtime environment is a virtual machine *(see Figure 2: 40).*

As per **Claim 27**, the rejection of **Claim 20** is incorporated; and Abbott further discloses:

- assigning a serialization identifier to each initialized object *(see Column 11: 46-50, "All instance objects can be found from the Reference Queue of the Java VM. In turn each object may contain references to other objects. The problem with these "object to object" references is that they refer to absolute locations (serialization identifier) inside the heap.").*

As per **Claim 28**, the rejection of **Claim 20** is incorporated; and Abbott further discloses:

- enumerating each object in a global scope and writing a serialized description of each said object *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 12: 6-10, "Thus the first item is to create a vector or table 510 having three columns. Each object in turn in the heap is then identified from the system Reference queue (enumerating each object), and added into the table 510, with one object entry per row of the table.").*

As per **Claim 29**, the rejection of **Claim 20** is incorporated; and Abbott further discloses:

- assigning a function ID to each function in the application *(see Column 16: 9-15, "... a table is created containing: the name of the DLL (function ID); its load location (memory address); and the NativeLibrary class association (in the case of DLLs containing JNI code).")*.

As per **Claim 30**, the rejection of **Claim 29** is incorporated; and Abbott further discloses:

- creating a function ID table associating each function ID with a function call *(see Column 16: 9-15, "... a table (function ID table) is created containing: the name of the DLL (function ID); its load location (memory address); and the NativeLibrary class association (in the case of DLLs containing JNI code).")*.

As per **Claim 31**, the rejection of **Claim 29** is incorporated; and Abbott further discloses:

- assigning function identifiers to functions within closures *(see Column 16: 2-7, "In order to save the state associated with such DLLs, all loaded DLLs are recorded with their base addresses (function identifiers), and if they contain JNI code, all references to them from class loaders and NativeLibrary classes (functions within closures) are also recorded (NativeLibrary classes provide the mechanism for a Java application to access the DLLs).")*.

As per **Claim 33**, Abbott discloses:

- requesting an application from an application source server *(see Column 3: 57-60, "In addition, the snapshot can be transmitted over a network, thereby allowing resumption on another system, for example for diagnostic purposes.")*;

- receiving object code of a serialized description of the application from the application source server, the object code including instructions for replicating a runtime memory state of the application *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)." and 57-60, "In addition, the snapshot can be transmitted over a network, thereby allowing resumption on another system, for example for diagnostic purposes."; Column 18: 13-16, "The reloading of the application is performed in stages. The various components of the Java VM are provided with restore commands to restart themselves using the saved information (instructions for replicating a runtime memory state of the application).")*; and

- loading the object code received from the application source server into a runtime environment *(see Column 3: 57-60, "In addition, the snapshot can be transmitted over a network, thereby allowing resumption on another system, for example for diagnostic purposes."; Column 18: 10-13, "In the preferred embodiment, the saved state is loaded (loading the object code) by a special "javaload" tool, which loads the stored application into a Java VM, bypassing all the initialisation normally associated with bringing up a Java VM.")*.

As per **Claim 36**, the rejection of **Claim 33** is incorporated; and Abbott further discloses:

- executing the object code *(see Column 18: 14-16, "The various components of the Java VM are provided with restore commands to restart themselves using the saved information.")*.

Claim Rejections - 35 USC § 103

23. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

24. **Claims 1, 2, 4, 6-8, 11-17, 19, 38, 39, 41, 42, 44, 45, 47-58, and 60-62** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Abbott** in view of **US 5,710,930 (hereinafter "Laney")**.

As per **Claim 1**, Abbott discloses:

- creating a serialized representation of application objects in a runtime environment *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 9: 10-20, "... the preferred embodiment of the present invention provides a set of callable routines and tools that allow a user to store away the state of an initialised Java application. The stored application is available for subsequent reloading, but without the start-up cost of initialisation.")*;

- building an optimized object code file using the serialized representation *(see Column 18: 31-35, "The table of object lengths and reference chains is used to reconstruct the object references inside each saved object. Class objects are also loaded at this time and their method tables are restored. This may involve reloading and rebuilding of JIT-compiled code.")*; and

- loading the optimized object code file into to a new runtime environment *(see Column 3: 57-60, "In addition, the snapshot can be transmitted over a network, thereby allowing resumption on another system, for example for diagnostic purposes."; Column 18: 10-13, "In the preferred embodiment, the saved state is loaded by a special "javaload" tool, which loads the stored application into a Java VM, bypassing all the initialisation normally associated with bringing up a Java VM.")*.

However, Abbott does not disclose:

- wherein the optimized object code file replaces an initialization code compiled from an application source code.

Laney discloses a shutdown program replaces an initialization code stored in a boot sector of a mass storage device of a computer system *(see Column 3: 38-48, "The shutdown program then creates a new system initialization code (i.e., loader program of the operating system) that contains code to cause a system state to be restored from the designated area of the nonvolatile memory via the shutdown program. The shutdown program then replaces the system initialization code currently stored in a boot sector of a mass storage device of the computer system with the new initialization code.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Laney into the teaching of Abbott to include wherein the optimized object code file replaces an initialization code compiled from an application source code. The modification would be obvious because one of ordinary skill in the art would be motivated to avoid reloading all the initialization normally associated with running a Java VM *(see Laney – Column 3: 60-64)*.

As per **Claim 2**, the rejection of **Claim 1** is incorporated; and Abbott further discloses:

- reading from a runtime memory space a description of each object of a running computer application *(see Column 12: 6-10, "Thus the first item is to create a vector or table 510 having three columns. Each object in turn in the heap is then identified from the system Reference queue, and added into the table 510, with one object entry per row of the table.")*.

As per **Claim 4**, the rejection of **Claim 2** is incorporated; and Abbott further discloses:

- wherein the runtime environment is a virtual machine *(see Figure 2: 40)*.

As per **Claim 6**, the rejection of **Claim 1** is incorporated; and Abbott further discloses:

- identifying each object of a running computer application by a unique identifier *(see Column 12: 10-15, "The first column 511 is used to hold the identifier or reference number of an object, and the third column 513 is used to hold the length of the object. The actual contents of the object can now be copied over into the save file for the snapshot (not shown in FIG. 5).")*.

As per **Claim 7**, the rejection of **Claim 6** is incorporated; and Abbott further discloses:

- detaching each object from an object hierarchy and creating a description of each slot in said object *(see Column 12: 29-36, "Therefore, as shown in the sequence of diagrams FIG. 5A, 5B, and 5C, the heap is scanned to detect all inter-object references. In the example of FIG. 5A, two such references are present, one 532A from object A to object S, and one 533A from*

object F to object S. As each object reference is found, a linked list is built up for that object, which is headed from the second column 512 of the vector table 510.").

As per **Claim 8**, the rejection of **Claim 6** is incorporated; and Abbott further discloses:

- providing the serialized representation directly to the building step *(see Column 18: 10-14, "... the saved state is loaded by a special "javaload" tool, which loads the stored application into a Java VM, bypassing all the initialisation normally associated with bringing up a Java VM. The reloading of the application is performed in stages.").*

As per **Claim 11**, the rejection of **Claim 1** is incorporated; and Abbott further discloses:

- assigning a serialization identifier to each object *(see Column 11: 46-50, "All instance objects can be found from the Reference Queue of the Java VM. In turn each object may contain references to other objects. The problem with these "object to object" references is that they refer to absolute locations (serialization identifier) inside the heap.").*

As per **Claim 12**, the rejection of **Claim 1** is incorporated; and Abbott further discloses:

- developing the computer application in an interpreted language *(see Column 7: 28-33, "Another component of the Java VM is the interpreter 156, which is responsible for reading in Java byte code from loaded classes, and converting this into machine instructions for the relevant platform.").*

As per **Claim 13**, the rejection of **Claim 6** is incorporated; and Abbott further discloses:

- assigning a function ID to each function in the computer application *(see Column 16: 9-15, "... a table is created containing: the name of the DLL (function ID); its load location (memory address); and the NativeLibrary class association (in the case of DLLs containing JNI code).")*.

As per **Claim 14**, the rejection of **Claim 13** is incorporated; and Abbott further discloses:

- creating a function ID table associating each function ID with function code *(see Column 16: 9-15, "... a table (function ID table) is created containing: the name of the DLL (function ID); its load location (memory address); and the NativeLibrary class association (in the case of DLLs containing JNI code).")*.

As per **Claim 15**, the rejection of **Claim 13** is incorporated; and Abbott further discloses:

- assigning unique function identifiers to functions within closures *(see Column 16: 2-7, "In order to save the state associated with such DLLs, all loaded DLLs are recorded with their base addresses (function identifiers), and if they contain JNI code, all references to them from class loaders and NativeLibrary classes (functions within closures) are also recorded (NativeLibrary classes provide the mechanism for a Java application to access the DLLs).")*.

As per **Claim 16**, the rejection of **Claim 1** is incorporated; and Abbott further discloses:

- using a compiled version of application source code compiled prior to the creating step in combination with the serialized representation to build the optimized object code file *(see Column 4: 21-26, "The preferred embodiment provides the user with various options, as to*

whether certain actions should be performed prior to determining the current state of the components of the virtual machine. Examples of such actions include performing a garbage collection, and forcing compilation of at least some of the application."; Column 18: 31-35, "The table of object lengths and reference chains is used to reconstruct the object references inside each saved object. Class objects are also loaded at this time and their method tables are restored. This may involve reloading and rebuilding of JIT-compiled code.").

As per **Claim 17**, the rejection of **Claim 1** is incorporated; and Abbott further discloses:

- writing the serialized representation to a text file prior to said step of building *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)." and 55-57, "A particular benefit of this approach is where multiple, identical, VMs can all be rapidly launched from a single save file (text file).").*

As per **Claim 19**, the rejection of **Claim 1** is incorporated; and Abbott further discloses:

- wherein the step of creating is performed in a different runtime *(see Column 9: 55-58, "Thus the method commences with the start of the Java application whose state is to be saved (step 410), which in turn leads in standard fashion to the initialisation of the Java VM classes (step 420)." Note that the runtime whose state is to be saved is the old runtime environment (different runtime).).*

As per **Claim 38**, Abbott discloses:

- creating a serialized representation of application objects in a runtime environment *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 9: 10-20, "... the preferred embodiment of the present invention provides a set of callable routines and tools that allow a user to store away the state of an initialised Java application. The stored application is available for subsequent reloading, but without the start-up cost of initialisation.")*;

- building an object code file using the serialized representation of the application objects and a compiled object code associated with a computer application *(see Column 18: 31-35, "The table of object lengths and reference chains is used to reconstruct the object references inside each saved object. Class objects are also loaded at this time and their method tables are restored. This may involve reloading and rebuilding of JIT-compiled code.")*; and

- loading the optimized object code file into a new runtime environment *(see Column 3: 57-60, "In addition, the snapshot can be transmitted over a network, thereby allowing resumption on another system, for example for diagnostic purposes."; Column 18: 10-13, "In the preferred embodiment, the saved state is loaded by a special "javaload" tool, which loads the stored application into a Java VM, bypassing all the initialisation normally associated with bringing up a Java VM.")*.

However, Abbott does not disclose:

- wherein the optimized object code file replaces an initialization code compiled from an application source code.

Laney discloses a shutdown program replaces an initialization code stored in a boot sector of a mass storage device of a computer system *(see Column 3: 38-48, "The shutdown program then creates a new system initialization code (i.e., loader program of the operating system) that contains code to cause a system state to be restored from the designated area of the nonvolatile memory via the shutdown program. The shutdown program then replaces the system initialization code currently stored in a boot sector of a mass storage device of the computer system with the new initialization code.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Laney into the teaching of Abbott to include wherein the optimized object code file replaces an initialization code compiled from an application source code. The modification would be obvious because one of ordinary skill in the art would be motivated to avoid reloading all the initialization normally associated with running a Java VM *(see Laney – Column 3: 60-64)*.

As per **Claim 39**, the rejection of **Claim 38** is incorporated; and Abbott further discloses:

- reading from a runtime environment memory space a description of each object of a running application *(see Column 12: 6-10, "Thus the first item is to create a vector or table 510 having three columns. Each object in turn in the heap is then identified from the system Reference queue, and added into the table 510, with one object entry per row of the table.")*.

As per **Claim 41**, the rejection of **Claim 38** is incorporated; and Abbott further discloses:

- identifying each object of a running application by a unique identifier *(see Column 12: 10-15, "The first column 511 is used to hold the identifier or reference number of an object, and the third column 513 is used to hold the length of the object. The actual contents of the object can now be copied over into the save file for the snapshot (not shown in FIG. 5).")*.

As per **Claim 42**, the rejection of **Claim 41** is incorporated; and Abbott further discloses:

- writing a description to a text file and compiling the text file *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)." and 55-57, "A particular benefit of this approach is where multiple, identical, VMs can all be rapidly launched from a single save file (text file)."; Column 4: 21-26, "The preferred embodiment provides the user with various options, as to whether certain actions should be performed prior to determining the current state of the components of the virtual machine. Examples of such actions include performing a garbage collection, and forcing compilation of at least some of the application (compiling the text file).")*.

As per **Claim 44**, the rejection of **Claim 41** is incorporated; and Abbott further discloses:

- detaching each object from an object hierarchy and creating a description of each slot in said object *(see Column 12: 29-36, "Therefore, as shown in the sequence of diagrams FIG. 5A, 5B, and 5C, the heap is scanned to detect all inter-object references. In the example of FIG. 5A, two such references are present, one 532A from object A to object S, and one 533A from*

object F to object S. As each object reference is found, a linked list is built up for that object, which is headed from the second column 512 of the vector table 510.").

As per **Claim 45**, the rejection of **Claim 41** is incorporated; and Abbott further discloses:

- determining whether the object is a class *(see Column 11: 5-8, "... all loaded classes are held as objects 145 on the heap 140, and may contain absolute references to instance objects, which need to be encoded for storage.")*; and

- writing a serialized description of the class *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization).").*

As per **Claim 47**, the rejection of **Claim 39** is incorporated; and Abbott further discloses:

- assigning a serialization identifier to each object *(see Column 11: 46-50, "All instance objects can be found from the Reference Queue of the Java VM. In turn each object may contain references to other objects. The problem with these "object to object" references is that they refer to absolute locations (serialization identifier) inside the heap.").*

As per **Claim 48**, the rejection of **Claim 39** is incorporated; and Abbott further discloses:

- developing the computer application in an interpreted language *(see Column 7: 28-33, "Another component of the Java VM is the interpreter 156, which is responsible for reading in Java byte code from loaded classes, and converting this into machine instructions for the relevant platform.").*

As per **Claim 49**, the rejection of **Claim 41** is incorporated; and Abbott further discloses:

- assigning a function ID to each function in the computer application *(see Column 16: 9-15, "... a table is created containing: the name of the DLL (function ID); its load location (memory address); and the NativeLibrary class association (in the case of DLLs containing JNI code).")*.

As per **Claim 50**, the rejection of **Claim 49** is incorporated; and Abbott further discloses:

- creating a function ID table associating each function ID with function code *(see Column 16: 9-15, "... a table (function ID table) is created containing: the name of the DLL (function ID); its load location (memory address); and the NativeLibrary class association (in the case of DLLs containing JNI code).")*.

As per **Claim 51**, the rejection of **Claim 49** is incorporated; and Abbott further discloses:

- assigning function identifiers to functions within closures *(see Column 16: 2-7, "In order to save the state associated with such DLLs, all loaded DLLs are recorded with their base addresses (function identifiers), and if they contain JNI code, all references to them from class loaders and NativeLibrary classes (functions within closures) are also recorded (NativeLibrary classes provide the mechanism for a Java application to access the DLLs).")*.

As per **Claim 52**, Abbott discloses:

- compiling first object code for an application *(see Column 4: 21-26, "The preferred embodiment provides the user with various options, as to whether certain actions should be performed prior to determining the current state of the components of the virtual machine. Examples of such actions include performing a garbage collection, and forcing compilation of at least some of the application.")*;

- loading the application into a first runtime environment *(see Column 5: 63-67, "The middleware also includes Java programming which acts to cause transactions as Java applications 50 to run on top of the Java VM 40.")*;

- creating a serialized representation of a memory space in said first runtime environment *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 9: 10-20, "... the preferred embodiment of the present invention provides a set of callable routines and tools that allow a user to store away the state of an initialised Java application. The stored application is available for subsequent reloading, but without the start-up cost of initialisation.")*;

- building a second object code using said serialized representation *(see Column 18: 31-35, "The table of object lengths and reference chains is used to reconstruct the object references inside each saved object. Class objects are also loaded at this time and their method tables are restored. This may involve reloading and rebuilding of JIT-compiled code.")*; and

- loading said second object code into a second runtime environment *(see Column 3: 57-60, "In addition, the snapshot can be transmitted over a network, thereby allowing resumption on another system, for example for diagnostic purposes."; Column 18: 10-13, "In*

the preferred embodiment, the saved state is loaded by a special "javaload" tool, which loads the stored application into a Java VM, bypassing all the initialisation normally associated with bringing up a Java VM.").

However, Abbott does not disclose:

- wherein the second object code replaces an initialization code compiled from an application source code.

Laney discloses a shutdown program replaces an initialization code stored in a boot sector of a mass storage device of a computer system *(see Column 3: 38-48, "The shutdown program then creates a new system initialization code (i.e., loader program of the operating system) that contains code to cause a system state to be restored from the designated area of the nonvolatile memory via the shutdown program. The shutdown program then replaces the system initialization code currently stored in a boot sector of a mass storage device of the computer system with the new initialization code.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Laney into the teaching of Abbott to include wherein the second object code replaces an initialization code compiled from an application source code. The modification would be obvious because one of ordinary skill in the art would be motivated to avoid reloading all the initialization normally associated with running a Java VM *(see Laney – Column 3: 60-64).*

As per **Claim 53**, the rejection of **Claim 52** is incorporated; and Abbott further discloses:

- wherein the step of compiling is performed on an interpreted language application *(see Column 7: 28-33, "Another component of the Java VM is the interpreter 156, which is responsible for reading in Java byte code from loaded classes, and converting this into machine instructions for the relevant platform.").*

As per **Claim 54**, the rejection of **Claim 52** is incorporated; and Abbott further discloses:

- wherein the step of creating is performed by calling at least one function from said first runtime environment *(see Column 9: 55-58, "Thus the method commences with the start of the Java application whose state is to be saved (step 410), which in turn leads in standard fashion to the initialisation of the Java VM classes (step 420).").*

As per **Claim 55**, the rejection of **Claim 52** is incorporated; and Abbott further discloses:

- wherein the step of creating is performed in the same runtime environment as said application *(see Column 9: 55-58, "Thus the method commences with the start of the Java application whose state is to be saved (step 410), which in turn leads in standard fashion to the initialisation of the Java VM classes (step 420).").*

As per **Claim 56**, the rejection of **Claim 52** is incorporated; and Abbott further discloses:

- wherein the step of creating is performed in a different runtime environment from said application *(see Column 9: 55-58, "Thus the method commences with the start of the Java application whose state is to be saved (step 410), which in turn leads in standard fashion to the*

initialisation of the Java VM classes (step 420)." Note that the runtime whose state is to be saved is the old runtime environment (different runtime).).

As per **Claim 57**, the rejection of **Claim 52** is incorporated; and Abbott further discloses:

- executing portions of the application marked for execution prior to said creating step *(see Column 10: 16-21, "... it may be desirable to perform some relatively trivial operation on the classes, such as accessing a class variable, prior to the save step (step 440), in order to ensure that the generic initialisations are indeed completed before the save operation.").*

As per **Claim 58**, Abbott discloses:

- receiving from a runtime environment a serialized representation of objects in a memory space of the runtime environment *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 9: 10-20, "... the preferred embodiment of the present invention provides a set of callable routines and tools that allow a user to store away the state of an initialised Java application. The stored application is available for subsequent reloading, but without the start-up cost of initialisation.")*; and

- building an optimized object code file using the serialized representation and a compiled object code file used to create the memory space *(see Column 18: 31-35, "The table of object lengths and reference chains is used to reconstruct the object references inside each saved object. Class objects are also loaded at this time and their method tables are restored. This may involve reloading and rebuilding of JIT-compiled code.").*

However, Abbott does not disclose:

- wherein the optimized object code file replaces an initialization code compiled from an application source code.

Laney discloses a shutdown program replaces an initialization code stored in a boot sector of a mass storage device of a computer system *(see Column 3: 38-48, "The shutdown program then creates a new system initialization code (i.e., loader program of the operating system) that contains code to cause a system state to be restored from the designated area of the nonvolatile memory via the shutdown program. The shutdown program then replaces the system initialization code currently stored in a boot sector of a mass storage device of the computer system with the new initialization code.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Laney into the teaching of Abbott to include wherein the optimized object code file replaces an initialization code compiled from an application source code. The modification would be obvious because one of ordinary skill in the art would be motivated to avoid reloading all the initialization normally associated with running a Java VM *(see Laney – Column 3: 60-64).*

As per **Claim 60**, the rejection of **Claim 58** is incorporated; and Abbott further discloses:

- initializing a serialization process in a separate memory space to create said serialized representation *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization).").*

As per **Claim 61**, the rejection of **Claim 58** is incorporated; and Abbott further discloses:

- initializing a serialization process in the runtime environment to create said serialized representation *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization).")*.

As per **Claim 62**, Abbott discloses:

- creating a serialized representation of application objects in a runtime environment *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 9: 10-20, "... the preferred embodiment of the present invention provides a set of callable routines and tools that allow a user to store away the state of an initialised Java application. The stored application is available for subsequent reloading, but without the start-up cost of initialisation.")*;

- building an optimized object code file using the serialized representation *(see Column 18: 31-35, "The table of object lengths and reference chains is used to reconstruct the object references inside each saved object. Class objects are also loaded at this time and their method tables are restored. This may involve reloading and rebuilding of JIT-compiled code.")*; and

- loading the optimized object code file into a new runtime environment via the network *(see Column 3: 57-60, "In addition, the snapshot can be transmitted over a network, thereby allowing resumption on another system, for example for diagnostic purposes."; Column*

18: 10-13, "In the preferred embodiment, the saved state is loaded by a special "javaload" tool, which loads the stored application into a Java VM, bypassing all the initialisation normally associated with bringing up a Java VM.").

However, Abbott does not disclose:

- wherein the optimized object code file replaces an initialization code compiled from an application source code.

Laney discloses a shutdown program replaces an initialization code stored in a boot sector of a mass storage device of a computer system *(see Column 3: 38-48, "The shutdown program then creates a new system initialization code (i.e., loader program of the operating system) that contains code to cause a system state to be restored from the designated area of the nonvolatile memory via the shutdown program. The shutdown program then replaces the system initialization code currently stored in a boot sector of a mass storage device of the computer system with the new initialization code.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Laney into the teaching of Abbott to include wherein the optimized object code file replaces an initialization code compiled from an application source code. The modification would be obvious because one of ordinary skill in the art would be motivated to avoid reloading all the initialization normally associated with running a Java VM *(see Laney – Column 3: 60-64).*

25. **Claims 3, 40, 63, and 64** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Abbott** in view of **Laney** as applied to Claims 1, 38, and 62 above, and further in view of **US 2004/0044989 (hereinafter "Vachuska")**.

As per **Claim 3**, the rejection of **Claim 1** is incorporated; however, Abbott and Laney do not disclose:

- enumerating a description of each object of the computer application using reflection.

Vachuska discloses:

- enumerating a description of each object of the computer application using reflection *(see Paragraph [0024], "The package 'java.lang.reflect' provides classes and interfaces for obtaining reflective information about classes and objects.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Vachuska into the teaching of Abbott to include enumerating a description of each object of the computer application using reflection. The modification would be obvious because one of ordinary skill in the art would be motivated to make it possible to examine the structure of the object *(see Vachuska – Paragraph [0004])*.

As per **Claim 40**, the rejection of **Claim 38** is incorporated; however, Abbott and Laney do not disclose:

- enumerating a description of each object of the computer application using reflection.

Vachuska discloses:

- enumerating a description of each object of the computer application using reflection *(see Paragraph [0024], "The package 'java.lang.reflect' provides classes and interfaces for obtaining reflective information about classes and objects.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Vachuska into the teaching of Abbott to include enumerating a description of each object of the computer application using reflection. The modification would be obvious because one of ordinary skill in the art would be motivated to make it possible to examine the structure of the object *(see Vachuska – Paragraph [0004])*.

As per **Claim 63**, the rejection of **Claim 62** is incorporated; however, Abbott and Laney do not disclose:

- enumerating a description of each object of the application using reflection.

Vachuska discloses:

- enumerating a description of each object of the application using reflection *(see Paragraph [0024], "The package 'java.lang.reflect' provides classes and interfaces for obtaining reflective information about classes and objects.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Vachuska into the teaching of Abbott to include enumerating a description of each object of the application using reflection. The modification would be obvious because one of ordinary skill in the art would be motivated to make it possible to examine the structure of the object *(see Vachuska – Paragraph [0004])*.

As per **Claim 64**, the rejection of **Claim 63** is incorporated; and Abbott further discloses:

- wherein the new runtime environment is a virtual machine *(see Figure 2: 40)*.

26. **Claims 5, 9, 10, 43, and 46** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Abbott** in view of **Laney** as applied to Claims 1, 4, 39, and 41 above, and further in view of **US 2003/0195923 (hereinafter "Bloch")**.

As per **Claim 5**, the rejection of **Claim 4** is incorporated; however, Abbott and Laney do not disclose:

- wherein the virtual machine is a presentation renderer.

Bloch discloses:

- wherein the virtual machine is a presentation renderer *(see Paragraph [0032], "In one embodiment, client Presentation Renderer 14 is a Macromedia Flash Player embedded in a web client as a plug-in.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Bloch into the teaching of Abbott to include wherein the virtual machine is a presentation renderer. The modification would be obvious because one of ordinary skill in the art would be motivated to access Internet applications and entertainment *(see Bloch – Paragraph [0006])*.

As per **Claim 9**, the rejection of **Claim 1** is incorporated; however, Abbott and Laney do not disclose:

- wherein the serialized representation of application objects in a runtime environment is written in an Extensible Markup Language data format.

Bloch discloses:

- wherein the serialized representation of application objects in a runtime environment is written in an Extensible Markup Language data format *(see Paragraph [0030], "In one environment, the mark-up language description is an XML-based language that is designed specifically for describing an application's user interface, along with the connection of that user-interface to various data sources and/or web services.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Bloch into the teaching of Abbott to include wherein the serialized representation of application objects in a runtime environment is written in an Extensible Markup Language data format. The modification would be obvious because one of ordinary skill in the art would be motivated to facilitate the sharing of data across different information systems, particularly via the Internet.

As per **Claim 10**, the rejection of **Claim 9** is incorporated; and Abbott further discloses:

- storing a file prior to said step of building *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization).").*

However, Abbott and Laney do not disclose:

- a markup language file.

Bloch discloses:

- a markup language file *(see Paragraph [0030], "In one environment, the mark-up language description is an XML-based language that is designed specifically for describing an application's user interface, along with the connection of that user-interface to various data sources and/or web services.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Bloch into the teaching of Abbott to include a markup language file. The modification would be obvious because one of ordinary skill in the art would be motivated to facilitate the sharing of data across different information systems, particularly via the Internet.

As per **Claim 43**, the rejection of **Claim 41** is incorporated; and Abbott further discloses:

- writing a description to a file and compiling a file *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 4: 21-26, "The preferred embodiment provides the user with various options, as to whether certain actions should be performed prior to determining the current state of the components of the virtual machine. Examples of such actions include performing a garbage collection, and forcing compilation of at least some of the application.")*.

However, Abbott and Laney do not disclose:

- an Extensible Markup Language file and a markup language file.

Bloch discloses:

- an Extensible Markup Language file and a markup language file *(see Paragraph [0030], "In one environment, the mark-up language description is an XML-based language that is designed specifically for describing an application's user interface, along with the connection of that user-interface to various data sources and/or web services.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Bloch into the teaching of Abbott to include an Extensible Markup Language file and a markup language file. The modification would be obvious because one of ordinary skill in the art would be motivated to facilitate the sharing of data across different information systems, particularly via the Internet.

As per **Claim 46**, the rejection of **Claim 39** is incorporated; however, Abbott and Laney do not disclose:

- wherein the serialized representation of application objects in a runtime environment is written in an Extensible Markup Language data format.

Bloch discloses:

- wherein the serialized representation of application objects in a runtime environment is written in an Extensible Markup Language data format *(see Paragraph [0030], "In one environment, the mark-up language description is an XML-based language that is designed specifically for describing an application's user interface, along with the connection of that user-interface to various data sources and/or web services.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Bloch into the teaching of Abbott to include

wherein the serialized representation of application objects in a runtime environment is written in an Extensible Markup Language data format. The modification would be obvious because one of ordinary skill in the art would be motivated to facilitate the sharing of data across different information systems, particularly via the Internet.

27. **Claims 24, 26, and 35** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Abbott** in view of **Bloch**.

As per **Claim 24**, the rejection of **Claim 22** is incorporated; and Abbott further discloses:

- writing the description to a file and providing the file to the rebuilder *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 18: 10-14, "... the saved state is loaded by a special "javaload" tool, which loads the stored application into a Java VM, bypassing all the initialisation normally associated with bringing up a Java VM. The reloading of the application is performed in stages.").*

However, Abbott does not disclose:

- an Extensible Markup Language file.

Bloch discloses:

- an Extensible Markup Language file *(see Paragraph [0030], "In one environment, the mark-up language description is an XML-based language that is designed specifically for describing an application's user interface, along with the connection of that user-interface to various data sources and/or web services.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Bloch into the teaching of Abbott to include an Extensible Markup Language file. The modification would be obvious because one of ordinary skill in the art would be motivated to facilitate the sharing of data across different information systems, particularly via the Internet.

As per **Claim 26**, the rejection of **Claim 25** is incorporated; however, Abbott does not disclose:

- wherein the virtual machine is a presentation renderer.

Bloch discloses:

- wherein the virtual machine is a presentation renderer *(see Paragraph [0032], "In one embodiment, client Presentation Renderer 14 is a Macromedia Flash Player embedded in a web client as a plug-in.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Bloch into the teaching of Abbott to include wherein the virtual machine is a presentation renderer. The modification would be obvious because one of ordinary skill in the art would be motivated to access Internet applications and entertainment *(see Bloch – Paragraph [0006])*.

As per **Claim 35**, the rejection of **Claim 33** is incorporated; however, Abbott does not disclose:

- wherein the runtime memory state is of a presentation renderer.

Bloch discloses:

- wherein the runtime memory state is of a presentation renderer *(see Paragraph [0032], "In one embodiment, client Presentation Renderer 14 is a Macromedia Flash Player embedded in a web client as a plug-in.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Bloch into the teaching of Abbott to include wherein the runtime memory state is of a presentation renderer. The modification would be obvious because one of ordinary skill in the art would be motivated to access Internet applications and entertainment *(see Bloch – Paragraph [0006])*.

28. **Claim 34** is rejected under 35 U.S.C. 103(a) as being unpatentable over **Abbott** in view of **US 5,987,256 (hereinafter "Wu")**.

As per **Claim 34**, the rejection of **Claim 33** is incorporated; however, Abbott does not disclose:

- wherein the object code includes media assets.

Wu discloses:

- wherein the object code includes media assets *(see Column 19: 35-37, "... a standard object file, such as an HTML or JAVA image, is input online 1300 to a compiler 1301 which runs on a standard computer 1302.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Wu into the teaching of Abbott to include

wherein the object code includes media assets. The modification would be obvious because one of ordinary skill in the art would be motivated to execute various media contents.

29. **Claim 65** is rejected under 35 U.S.C. 103(a) as being unpatentable over **Abbott** in view of **Laney** and **Vachuska** as applied to Claim 64 above, and further in view of **Bloch**.

As per **Claim 65**, the rejection of **Claim 64** is incorporated; however, Abbott, Laney, and Vachuska do not disclose:

- wherein the virtual machine is a presentation renderer.

Bloch discloses:

- wherein the virtual machine is a presentation renderer *(see Paragraph [0032], "In one embodiment, client Presentation Renderer 14 is a Macromedia Flash Player embedded in a web client as a plug-in.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Bloch into the teaching of Abbott to include wherein the virtual machine is a presentation renderer. The modification would be obvious because one of ordinary skill in the art would be motivated to access Internet applications and entertainment *(see Bloch – Paragraph [0006])*.

Response to Arguments

30. Applicant's arguments with respect to Claims 1, 38, 52, 58, and 62 regarding replacing the initialization code have been considered, but are moot in view of the new ground(s) of rejection.

In the Remarks, Applicant argues:

a) Unlike the method recited in claim 1, Abbott does not build a new object code ("optimized object code file") replacing the initialization code and subsequently load the new object code in a new runtime environment. As discussed above, Abbott simply takes a "snapshot" of the state if the initialized application and recreates the application, component by component.

Examiner's response:

a) Examiner disagrees. Applicant's arguments are not persuasive for at least the following reasons:

First, Abbott clearly discloses "building an optimized object code file using the serialized representation" *(see Column 18: 31-35, "The table of object lengths and reference chains is used to reconstruct the object references inside each saved object. Class objects are also loaded at this time and their method tables are restored. This may involve reloading and rebuilding of JIT-compiled code.")*. Thus, during the restoration of the application using the saved information (serialized representation), class objects and their method tables are being restored by reloading and rebuilding JIT-compiled code (optimized object code). Abbott also clearly discloses "loading

the optimized object code file into to a new runtime environment" *(see Column 3: 57-60, "In addition, the snapshot can be transmitted over a network, thereby allowing resumption on another system, for example for diagnostic purposes."; Column 18: 10-13, "In the preferred embodiment, the saved state is loaded by a special "javaload" tool, which loads the stored application into a Java VM, bypassing all the initialisation normally associated with bringing up a Java VM.")*. Note that the snapshot is loaded by the "javaload" tool, bypassing all the initialization, to restore the VM state so the suspended application can resume execution. Also, note that the VM state can be restored on another computer system.

Second, the claims recite only "object code file" with no further clarification on the claim scope of the limitation "object code file" as intended by the Applicant to cover. Thus, as the claims are interpreted as broadly as their terms reasonably allow (see MPEP § 2111.01 I), the interpretation of a broad limitation of "object code file" as JIT-compiled code and the like by one of ordinary skill in the art is considered to be reasonable by its plain meaning.

In the Remarks, Applicant argues:

b) For at least the same reasons discussed above with regard to claim 1, the method recited in claim 20 is not anticipated by Abbott. Namely, Abbott does not disclose a serialized representation that "includes memory relations between application objects and a serialization identifier associated with each application object." Therefore, Applicants respectfully request that the Examiner remove this rejection.

Examiner's response:

b) Examiner disagrees. Applicant's arguments are not persuasive for at least the following reasons:

First, in response to Applicant's argument that the references fail to show certain features of Applicant's invention, it is noted that the features upon which Applicant relies (*i.e.*, a serialized representation that "includes memory relations between application objects and a serialization identifier associated with each application object") are not recited in the rejected claim(s). Although the claims are interpreted in light of the specification, limitations from the specification are not read into the claims. See *In re Van Geuns*, 988 F.2d 1181, 26 USPQ2d 1057 (Fed. Cir. 1993).

Second, Applicant's arguments fail to comply with 37 CFR 1.111(b) because they amount to a general allegation that the claims define a patentable invention without specifically pointing out how the language of the claims patentably distinguishes them from the references.

Third, Applicant's arguments relating to Claim 1 only concern with the particular limitation of "building an optimized object code file using the serialized representation, wherein the optimized object code file replaces the initialization code compiled from the application source code." Claim 20 does not recite such limitation and, therefore, Applicant's arguments with regard to Claim 1 do not apply to Claim 20.

In the Remarks, Applicant argues:

c) For at least the same reasons discussed above with regard to claim 1, the method recited in claim 33 is not anticipated by Abbott. Namely, Abbot does not disclose receiving an object

code from an application source server and subsequently loading the object code into a runtime environment. Therefore, Applicants respectfully request that the Examiner remove this rejection.

Examiner's response:

c) Examiner disagrees. Applicant's arguments are not persuasive for at least the following reasons:

First, Applicant's arguments fail to comply with 37 CFR 1.111(b) because they amount to a general allegation that the claims define a patentable invention without specifically pointing out how the language of the claims patentably distinguishes them from the references.

Second, Abbott clearly discloses "receiving object code of a serialized description of the application from the application source server" *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)." and 57-60, "In addition, the snapshot can be transmitted over a network, thereby allowing resumption on another system, for example for diagnostic purposes.")*. Note that the snapshot can be transmitted over a network (*e.g.*, servers and clients) by requesting and receiving data. Regarding the limitation of "loading the object code received from the application source server into a runtime environment," Examiner has addressed this limitation in the Examiner's response (a) above.

Third, Applicant's arguments relating to Claim 1 only concern with the particular limitation of "building an optimized object code file using the serialized representation, wherein the optimized object code file replaces the initialization code compiled from the application

source code." Claim 33 does not recite such limitation and, therefore, Applicant's arguments with regard to Claim 1 do not apply to Claim 33.

In the Remarks, Applicant argues:

d) For at least the same reasons discussed above with regard to claim 1, the method recited in claim 38 is not anticipated by Abbott. Namely, Abbot does not disclose building an "optimized object code file" and loading the file into a new runtime environment. Therefore, Applicants respectfully request that the Examiner remove this rejection.

Examiner's response:

d) Examiner has addressed Applicant's arguments in the Examiner's response (a) above.

In the Remarks, Applicant argues:

e) For at least the same reasons discussed above with regard to claim 1, the method recited in claim 52 is not anticipated by Abbott. Namely, Abbott does not disclose "building a second object code" that is then loaded into a "second runtime environment." Therefore, Applicants respectfully request that the Examiner remove this rejection.

Examiner's response:

e) Examiner has addressed Applicant's arguments in the Examiner's response (a) above.

In the Remarks, Applicant argues:

f) For at least the same reasons discussed above with regard to claim 1, the method recited in claim 58 is not anticipated by Abbott. Namely, Abbott does not disclose "building an optimized object code" that replaces the initialization code compiled from the application source code. Therefore, Applicants respectfully request that the Examiner remove this rejection.

Examiner's response:

f) Examiner has addressed Applicant's arguments in the Examiner's response (a) above.

In the Remarks, Applicant argues:

g) For at least the same reasons discussed above regarding claim 1, Applicants respectfully suggest that the method recited in claim 62 is not anticipated by Abbott. Namely, Abbott does not disclose "building an optimized object code" that replaces the initialization code compiled from the application source code. Therefore, Applicants request that the Examiner remove this rejection.

Examiner's response:

g) Examiner has addressed Applicant's arguments in the Examiner's response (a) above.

Note that Applicant did not traverse the Examiner's assertion of Official Notice with regard to Claims 5, 9, 10, 24, 26, 35, 43, 46, and 65. Therefore, the "old and well-known within the computing art" statement is taken to be admitted prior art because Applicant has failed to traverse the Examiner's assertion of Official Notice (see MPEP § 2144.03).

Conclusion

31. The prior art made of record and not relied upon is considered pertinent to Applicant's disclosure.

32. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Qing Chen whose telephone number is 571-270-1071. The Examiner can normally be reached on Monday through Thursday from 7:30 AM to 4:00 PM. The Examiner can also be reached on alternate Fridays.

If attempts to reach the Examiner by telephone are unsuccessful, the Examiner's supervisor, Wei Zhen, can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

/QC/
February 25, 2008

/MARY STEELMAN/
for /Mary Steelman/, Primary Examiner of Art Unit 2191